Clickseo.com

C Programming

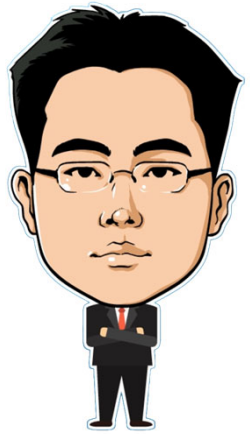
포인터

(Pointers)

Seo, Doo-Ok

Clickseo.com

clickseo@gmail.com



CC BY NC ND

목 차

CC BY NC ND

백문이불여일타(百聞而不如一打)

- 포인터의 이해
- 다양한 포인터

포인터의 이해

CC BY NC ND

백문이불여일타(百聞而不如一打)

- **포인터의 이해**
  - 포인터 변수
  - 포인터 연산
- 다양한 포인터

포인터 변수 (1/11)

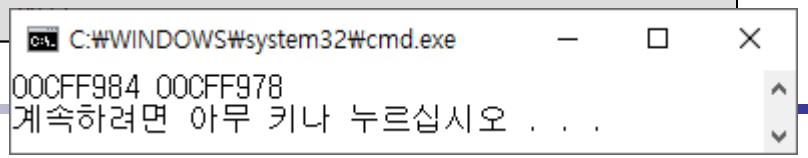
● 주소 연산자 (&)

❍ 변수와 배열 원소에만 적용한다.

- 산술식이나 상수에는 주소 연산자를 사용할 수 없다.
- 레지스터 변수 또한 주소 연산자를 사용할 수 없다.

```
&100;                   // 상수 에는 주소 연산자를 사용할 수 없다.
&(a+1)                  // 산술식 에는 주소 연산자를 사용할 수 없다.
register  a;            // 레지스터 변수 에는 주소 연산자를 사용할 수 없다.


int          a;
&a;
```

```
#include <stdio.h>
int main(void)
{
    int     a;
    int     b;

    printf("%p %p ₩n", &a, &b);

    return 0;
}
```



포인터 변수 (2/11)

- **포인터형 변수**(Pointer Variable)
  - 포인터
    - 메모리 주소에 대한 기호화 된 표현

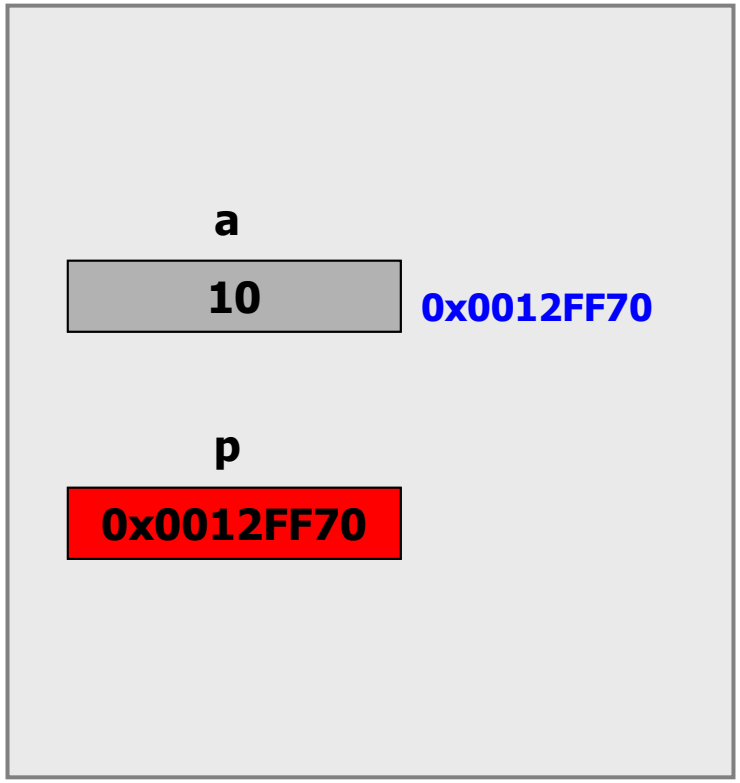
```
int *p;
```

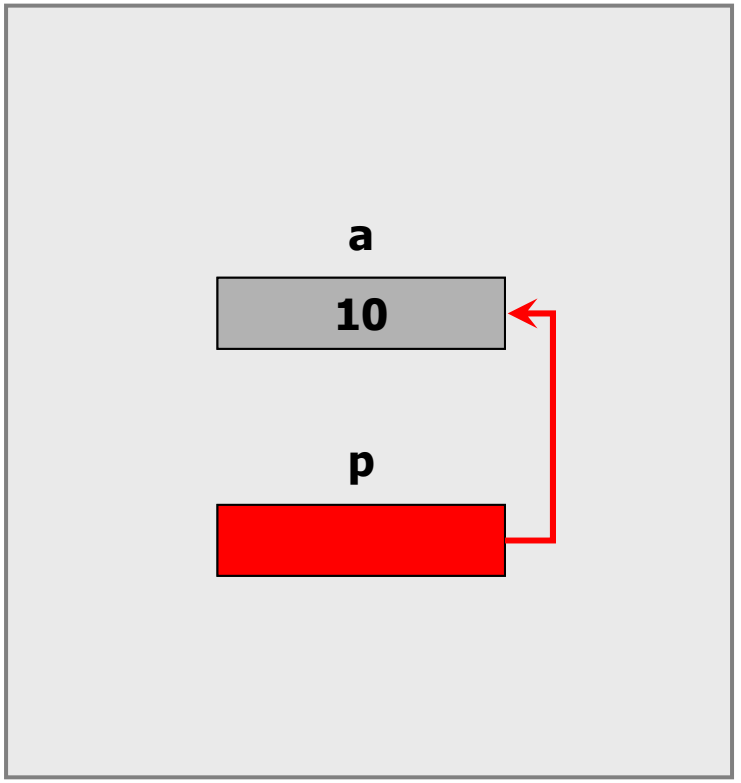
  - 포인터형 변수 : 메모리 주소를 를 저장할 수 있는 변수
  - 간접 연산자 (*) 또는 역 참조 연산자
    - 포인터 변수에 저장된 주소를 참조하여 대상체의 크기(자료형) 만큼 메모리를 간접 접근한다.

"포인터 변수는 메모리 주소 이외에는
어떠한 값도 저장하지 않는다는 것을 절대 잊으면 안 된다."

포인터 변수 (3/11)

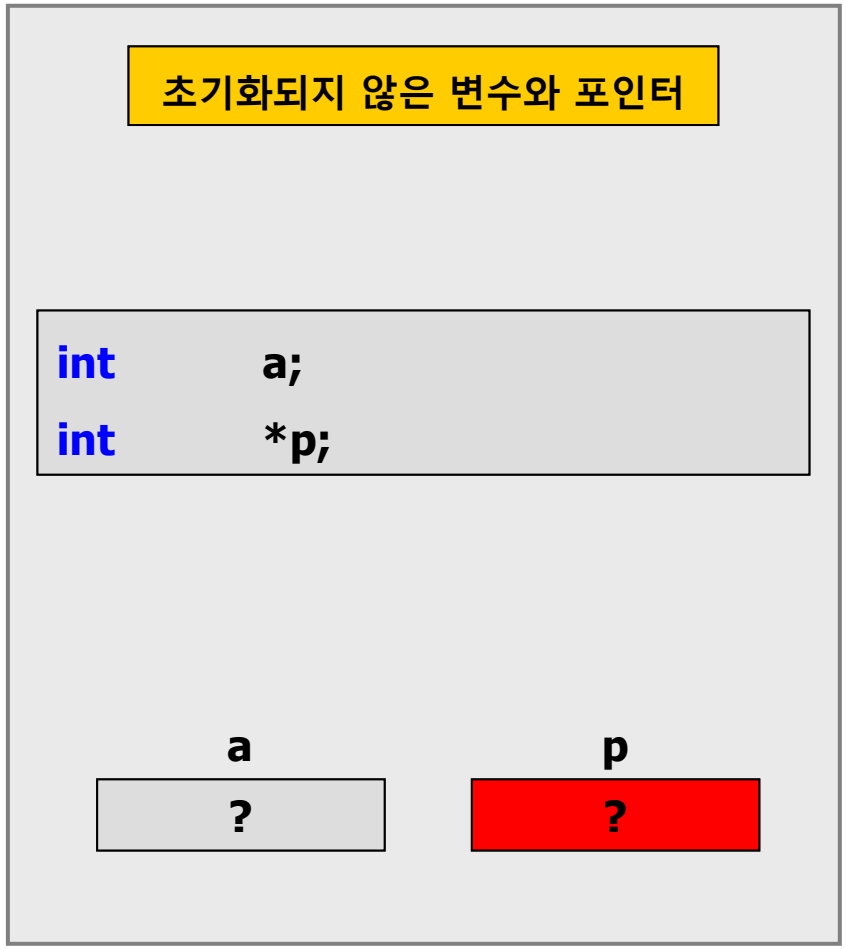
- **포인터형 변수** : 물리적/논리적 표현
  - 물리적인 표현과 논리적인 표현

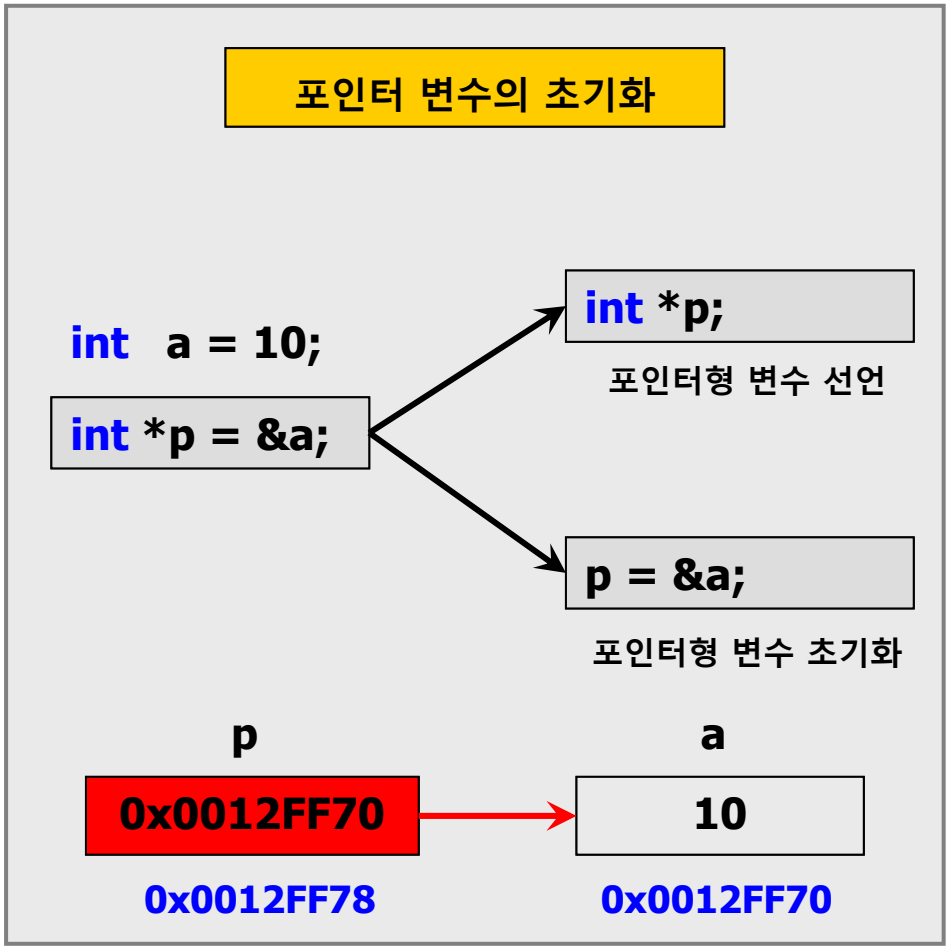




포인터 변수 (4/11)

- **포인터형 변수** : 선언 및 초기화

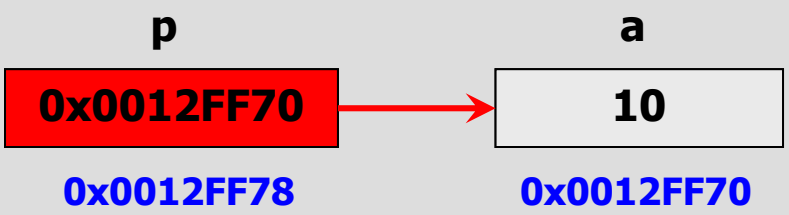




포인터 변수 (5/11)

- **포인터형 변수** : 간접 접근

```
// 일반 변수와 포인터형 변수 선언
int         a;
int         *p;

// 일반 변수와 포인터형 변수의 초기화
a = 10;
p = &a;
```

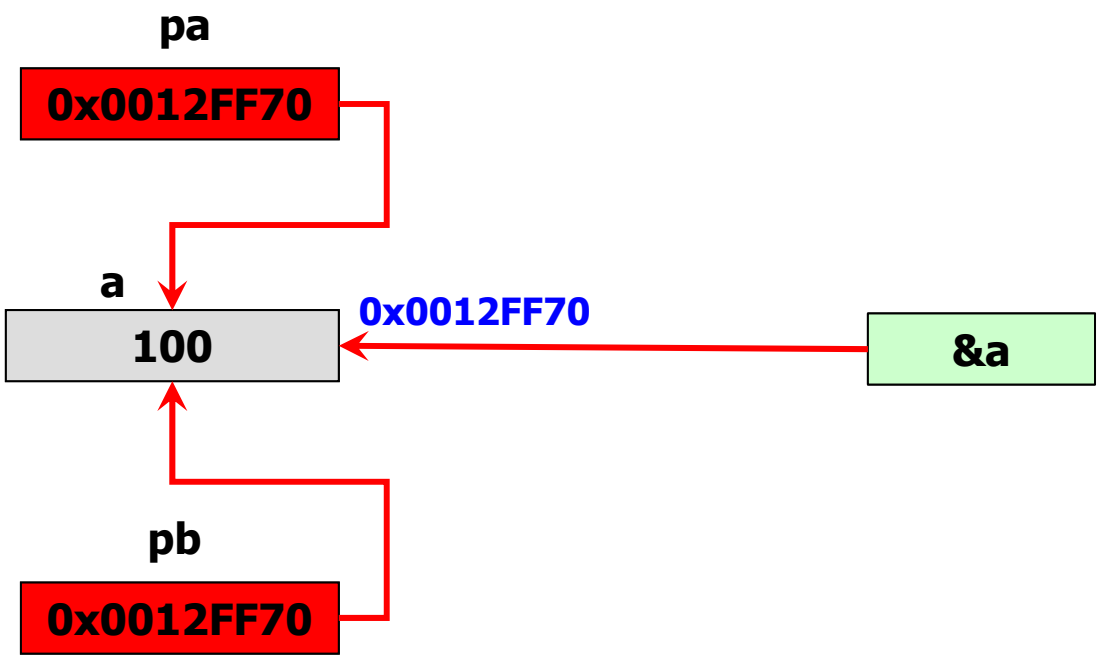



```
*p      a       10              // 변수 a 의 데이터 값
p       &a      0x0012FF70      // 변수 a 의 주소
```

```
a = a + 1;          a++;
*p = *p + 1;        (*p)++;
```

포인터 변수 (6/11)

- **포인터형 변수** : 다중 포인터
  - 변수에 대한 다중 포인터



포인터 변수 (7/11)

- **포인터형 변수에 왜 자료형을 지정하는가?**

```
// 컴파일러는 아래 문장을 어떻게 처리할까?
// 즉, 메모리에 몇 바이트 씩을 할당할까?

char      *pc;
int       *pi;
float     *pf;
double    *pd;
```

"포인터형 변수는 항상 4 bytes 메모리가 할당된다."

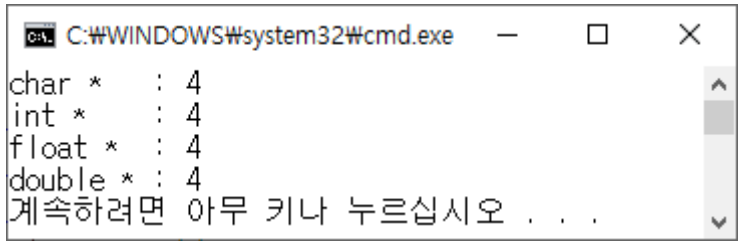
Data Type : unsigned int

"포인터형 변수의 자료형은 포인터 변수가 가리키는 대상체의 크기(자료형),
즉, 저장하고 있는 메모리 주소를 간접 접근하여
어떤 형태(자료형)로 접근(해석)할 것인가를 의미한다."

포인터 변수 (8/11)

예제 3-1 : 포인터형 변수의 메모리 할당 Visual Studio Community 2019

```
#include <stdio.h>
int main(void)
{
        char    *pc;
        int     *pi;
        float   *pf;
        double  *pd;

        printf("char *   : %d \n", sizeof(char *));
        printf("int *    : %d \n", sizeof(int *));
        printf("float *  : %d \n", sizeof(float *));
        printf("double * : %d \n", sizeof(double *));

        return 0;
}
```



포인터 변수 (9/11)

예제 3-2 : 포인터형 변수의 메모리 할당 GCC(GNU Compiler Collection)

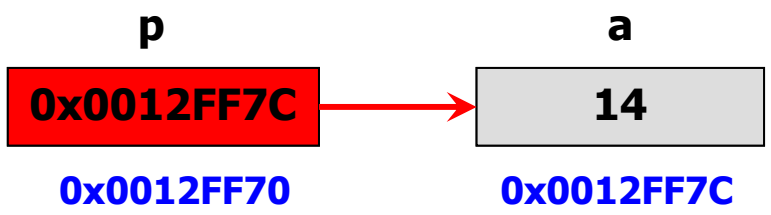
```c
#include <stdio.h>
int main(void)
{
        char    *pc;
        int     *pi;
        float   *pf;
        double  *pd;

        // warning: format '%d' expects argument of type 'int',
        // but argument 2 has type 'long unsigned int'
        // printf("char * : %d \n", sizeof(char *) );

        printf("char *   : %ld \n", sizeof(char *));
        printf("int *    : %ld \n", sizeof(int *));
        printf("float *  : %ld \n", sizeof(float *));
        printf("double * : %ld \n", sizeof(double *));

        return 0;
}
```

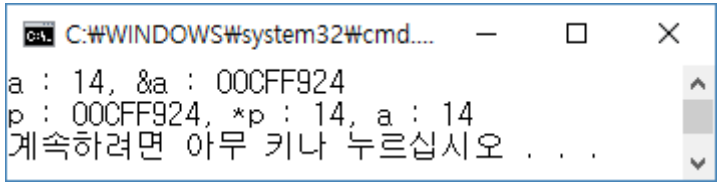
```
clickseo@clickseo-VirtualBox: ~/gcc
clickseo@clickseo-VirtualBox:~/gcc$ gcc pointer.c
clickseo@clickseo-VirtualBox:~/gcc$ ./a.out
char *   : 8
int *    : 8
float *  : 8
double * : 8
clickseo@clickseo-VirtualBox:~/gcc$
```

포인터 변수 (10/11)

예제 3-3 : 포인터형 변수의 선언 및 초기화 그리고 간접 접근

```
#include <stdio.h>
int main(void)
{
        int     a;
        int     *p;

        a = 14;
        p = &a;

        printf("a : %d, &a : %p\n", a, &a );
        printf("p : %p, *p : %d, a : %d \n", p, *p, a );

        return 0;
}
```

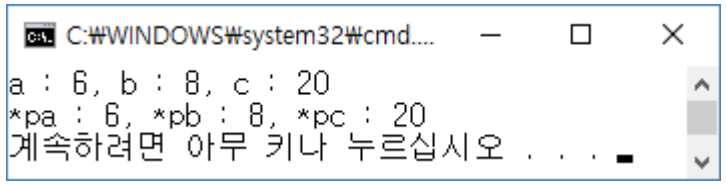




포인터 변수 (11/11)

예제 3-4 : 일반 변수의 직접 접근과 포인터형 변수의 간접 접근

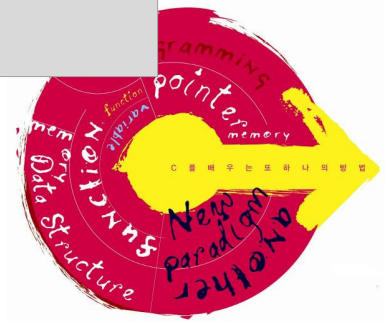
```
#include <stdio.h>
int main(void)
{
        int     a, b, c;
        int     *pa, *pb, *pc;

        a = 6;
        b = 2;

        pa = &b;
        pb = pa;
        pc = &c;

        pa = &a;
        *pb = 8;

        *pc = *pa;
        *pc = a + *pb + *&c;

        printf("a : %d, b : %d, c : %d \n",  a, b, c );
        printf("*pa : %d, *pb : %d, *pc : %d \n", *pa, *pb, *pc );

        return 0;
}
```



CC BY NC ND

포인터의 이해

포인터 연산



포인터 연산 (1/3)

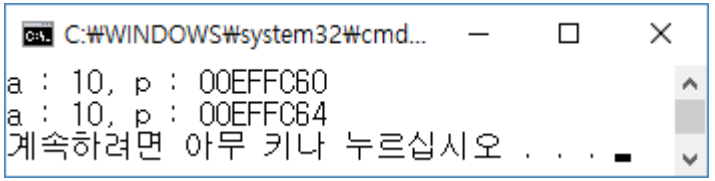
- **포인터 연산**
  - 연산의 대상 : 메모리 주소
    - 정수 연산만 가능
    - 사용 가능 연산자 : + , - , ++ , -- , > , >= , < , <= , == , !=
  - 포인터가 가리키는 대상체(자료형)의 크기 만큼 연산이 일어난다.

```
int         a = 10;
int         *p;

p  =  &a;

p++;                    // 실제로 증가되는 대상은?
```

포인터 연산 (2/3)

예제 3-5 : 포인터 연산 -- 증감 연산자

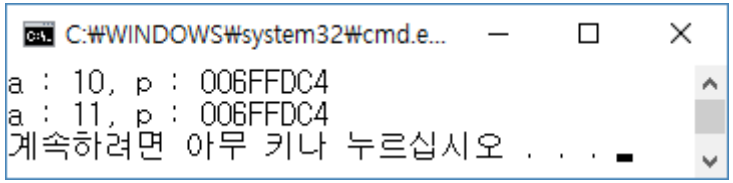
```
#include <stdio.h>
int main(void)
{
        int     a = 10;
        int     *p;

        p = &a;

        printf("a : %d, p : %p \n", a, p );

        p++;

        printf("a : %d, p : %p \n", a, p );

        return 0;
}
```



포인터 연산 (3/3)

예제 3-6 : 포인터 연산 -- 간접 연산자와 증감 연산자

```
#include <stdio.h>
int main(void)
{
        int     a = 10;
        int     *p;

        p = &a;

        printf("a : %d, p : %p \n", a, p );

        (*p)++;

        printf("a : %d, p : %p \n", a, p );

        return 0;
}
```



다양한 포인터

CC BY NC ND

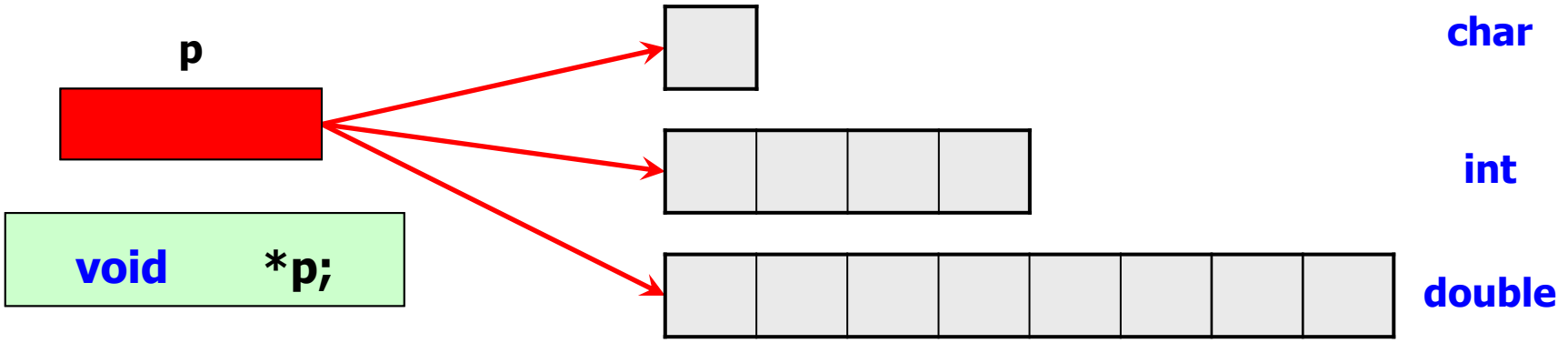
- 포인터의 이해

백문이불여일타(百聞而不如一打)

- **다양한 포인터**
  - void 포인터
  - NULL 포인터
  - 다중 포인터

void 포인터 (1/6)

- **void형 포인터**
  - 어떤 대상체(자료형)의 메모리 주소 든지 저장할 수 있는 포인터
    - 포인터 변수 선언 시 간접 접근 방법을 지정하지 않은 포인터
    - 간접 접근으로 대상체의 메모리를 접근 할 때는 반드시 형 변환이 이루어져야 한다.



void 포인터 (2/6)

- **void형 포인터 변수** : 다양한 자료형의 메모리 주소 저장

  1. 어떠한 형 변환 없이도 void형 포인터 변수에 메모리 주소의 저장이 가능하다.

```c
#include <stdio.h>
int main(void)
{
    char        c;
    int         i;
    float       f;
    double      d;

    void        *p;

    p = &c;
    p = &i;
    p = &f;
    p = &d;

    return 0;
}
```

```c
#include <stdio.h>
int main(void)
{
    char  c;
    int   i;

    int   *p;

    // warning C4133: '=' : incompatible types - from 'char *' to 'int *'
    p = &c;          //  p = (int *)&c;
    p = &i;

    return 0;
}
```

void 포인터 (3/6)

- **void형 포인터 변수** : 다양한 포인터의 값(메모리 주소) 저장

```c
#include <stdio.h>
int main(void)
{
    char        *pc;
    int         *pi;
    float       *pf;
    double      *pd;

    void        *p;

    p = pc;
    p = pi;
    p = pf;
    p = pd;

    return 0;
}
```

```c
#include <stdio.h>
int main(void)
{
    char  *pc;
    int   *pi;

    int   *p;

    // warning C4133: '=' : incompatible types - from 'char *' to 'int *'
    p = pc;             //          p = (int *)pc;
    p = pi;

    return 0;
}
```

void 포인터 (4/6)

- **void형 포인터 변수** : 포인터 변수와 형 변환

  2. void형 포인터 변수를 이용하여 일반 변수의 값을 읽을 때는 반드시 형 변환이 이루어져야 한다.

```
#include <stdio.h>
int main(void)
{
    char            c = 'A';
    int             i = 5;
    double          d = 0.7;

    void    *p;

    p = &c;
    printf("c : %c \n",   *(char *)p );
    p = &i;
    printf("i : %d \n",   *(int *)p );
    p = &d;
    printf("d : %.1f \n",   *(double *)p );

    return 0;
}
```

void 포인터 (5/6)

- **void형 포인터 변수** : 간접 접근과 대상체의 크기(자료형)

  3. 간접 연산자를 사용할 때는 포인터 변수가 저장하고 있는 메모리 주소의 대상체의 크기(자료형)과 일치해야 한다.

```
int       i = 5;

void      *p;

// 정수형 변수 i 의 메모리 주소를 포인터형 변수 p에 저장한다.
p = &i;

// 포인터형 변수 p 가 가리키고 있는 메모리 영역을 int 형식(4bytes)으로 읽어야 한다.
printf("i : %f \n", *(double *)p );                    // 무의미!!!
```

```
printf("i : %d₩n", *(int *) p ) ;
```

void 포인터 (6/6)

- **void형 포인터 변수** : 간접 접근과 증감 연산자

  4. **void형 포인터 변수에 ++, -- 를 사용할 때에는 항상 형 변환이 이루어져야 한다.**

     - ++ 나 -- 를 사용하려면 정확한 대상체의 크기(자료형)을 알아야 그 크기 만큼의 정확한 연산이 이루어진다. 정확한 대상체의 크기(자료형)을 명시하지 않으면, 컴파일러는 어떤 형태로 간접 접근할지를 알 수 없기 때문에 문제가 발생한다.

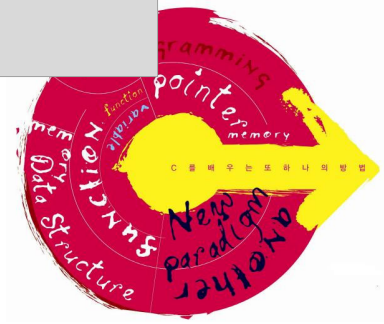
```
#include <stdio.h>
int main(void)
{
    int             i = 50;
    void            *p;

    p = &i;

    // *p = *p + 1;
    (*p)++;         // error C2036: 'void *' : unknown size

    return 0;
}
```

```
*(int *) p = *(int *) p + 1;

( *(int *) p )++;
```

CC BY NC ND

다양한 포인터

NULL 포인터, 다중 포인터



NULL 포인터

- **NULL 포인터**
  - 매크로 상수 : 널 포인터 상수

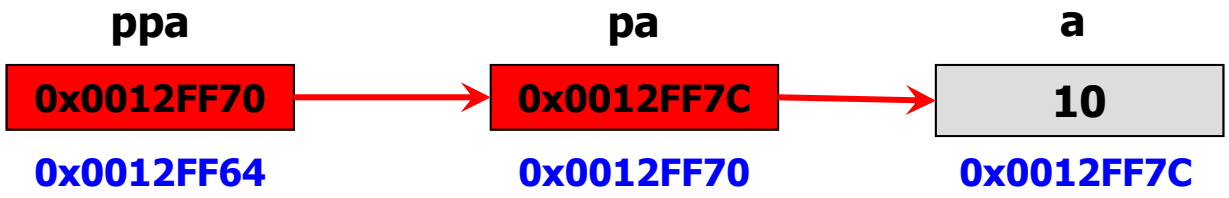
```
#define  NULL  (void *)0
```

```
#include <stdio.h>              // NULL
int main(void)
{
    // NULL 포인터
    int    *ptr = NULL;

    printf("ptr : %u", ptr );

    return 0;
}
```

```
C:₩WINDOWS₩system32₩cmd.exe
ptr : 0
계속하려면 아무 키나 누르십시오 . . .
```

다중 포인터 (1/2)

- **이중 포인터(Double Pointer)**
  - 포인터에 대한 포인터

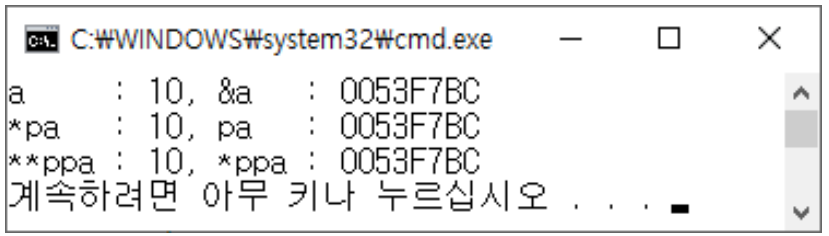



```
int       a;
int       *pa;
int       **ppa;

a = 10;
pa = &a;
ppa = &pa;

printf("%3d", a );
printf("%3d", *pa );
printf("%3d", **ppa );
```

다중 포인터 (2/2)

예제 3-7 : 이중 포인터

```
#include <stdio.h>
int main(void)
{
        int     a = 10;
        int     *pa;
        int     **ppa;

        pa = &a;
        ppa = &pa;

        printf("a     : %d, &a    : %p \n", a, &a );
        printf("*pa   : %d, pa    : %p \n", *pa, pa );
        printf("**ppa : %d, *ppa  : %p \n", **ppa, *ppa );

        return 0;
}
```



참고문헌

[1] 서두옥, 이동호(감수), (열혈강의)"또 하나의 C : 프로그래밍은 셀프입니다", 프리렉, 2012.

[2] Paul Deitel, Harvey Deitel, "C How to Program", Global Edition, 8/E, Pearson, 2016.

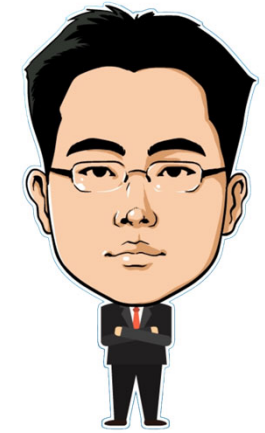
[3] SAMUEL P. HARBISON Ⅲ, GUY L. STEELE, "C 프로그래밍 언어, C : A Reference Manual", 5/E, Pearson Education Korea, 2005.

[4] Behrouz A. Forouzan, Richard F. Gilberg, 김진 외 7인 공역, "구조적 프로그래밍 기법을 위한 C", 도서출판 인터비젼, 2004.

[5] Brian W. Kernighan, Dennis M. Ritchie, 김석환 외 2인 공역, "The C Programming Language", 2/E, 대영사, 2004.

[6] 김일광, "C 프로그래밍 입문 : 프로그래밍을 모국어처럼 유창하게", 한빛미디어, 2004.

[7] 정재은, "다시 체계적으로 배우는 C 언어 포인터", 정보문화사, 2003.



이 강의자료는 저작권법에 따라 보호받는 저작물이므로 무단 전제와 무단 복제를 금지하며,
내용의 전부 또는 일부를 이용하려면 반드시 저작권자의 서면 동의를 받아야 합니다.

Copyright © Clickseo.com. All rights reserved.